VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Portace programátoru USBPICPROG na mikropočítač řady PIC24x

USBPICPROG Porting to a New PIC24x Microcontroller Line

2017 Lukáš Doležel

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání bakalářské práce

Student: **Lukáš Doležel**

Studijní program: B2647 Informační a komunikační technologie

Studijní obor: 2612R025 Informatika a výpočetní technika

Téma: Portace programátoru USBPICPROG na miktopočítač řady PIC24x
USBPICPROG Porting to a New PIC24x Microcontroller Line

Jazyk vypracování: čeština

Zásady pro vypracování:

Realizujte portaci otevřeného projektu USBPICPROG ze zastaralého mikrokontroléru PIC18F2250 na modernější platformu mikrokontrolerů PIC24Fxx. Nahraďte nutnost využívání knihovny USBLIB využitím standardního rozhraní - emulací sériového portu CDC.

1. Seznamte se s technickým řešením programátoru USBPICPROG. Zapojte si na kontaktním nepájivém poli funkční programátor.
2. Vyberte z řady mikrokontrolérů PIC24Fxx vhodný typ pro realizaci nové verze programátoru.
3. Zapojte na kontaktním nepájivém poli prototypové zařízení.
4. Navrhněte a realizujte úpravy programu mikrokontroléru pro nový typ mikrokontroléru.
5. Nahraďte komunikační rozhraní mezi PC a mikrokontrolérem emulátorem sériové linky a upravte dle potřeby programy na obou platformách.
6. Otestujte funkčnost a spolehlivost navrženého řešení.

Seznam doporučené odborné literatury:

[1] Open Source programátor USBPICPROG - http://www.usbpicprog.org
[2] Datové listy mikrokontrolérů Microchip - http://www.microchip.com

Formální náležitosti a rozsah bakalářské práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí bakalářské práce: **Ing. Petr Olivka, Ph.D.**

Datum zadání: 01.09.2015

Datum odevzdání: 28.04.2017

doc. Dr. Ing. Eduard Sojka
vedoucí katedry

prof. RNDr. Václav Snášel, CSc.
děkan fakulty

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 28. června 2017 .....

Souhlasím se zveřejněním této bakalářské práce dle požadavků čl. 26, odst. 9 Studijního a zkušebního řádu pro studium v bakalářských programech VŠB-TU Ostrava.

V Ostravě 28. června 2017 .....

Rád bych poděkoval vedoucímu bakalářské práce Ing. Petrovi Olivkovi, Ph.D. za odbornou pomoc a konzultace při vytváření této práce.

Abstrakt

Práce je zaměřena na portaci otevřeného projektu programátoru Usbpicprog na novější verzi mikrokontroléru, z důvodu větší dostupné paměti pro další rozšiřování ovládacího firmwaru a zároveň nahrazení komunikačního protokolu USB využívající knihovnu libusb, za komunikaci pomocí USB CDC, který využívá rozhraní USB, ale ke komunikaci využívá sériovou komunikaci.

Klíčová slova: usbpicprog, libusb, Communication Device Class, Microchip, PIC18, PIC24F

Abstract

The thesis is focused on porting the open source project of the programmer Usbpicprog to a newer version of the microcontroller because of the more available memory for further expansion of the control firmware while replacing the USB communication protocol using libusb libraries for USB CDC communication using USB interface but for communication using serial communication.

Key Words: usbpicprog, libusb, Communication Device Class, Microchip, PIC18, PIC24F

Obsah

Seznam použitých zkratek a symbolů

ACM	–	Abstract Control Model
API	–	Application Programming Interface
CCP	–	Capture/Compare/PWM
CDC	–	Comunication Device Class
ICSP	–	In-Circuit Serial Programming
ISP	–	In-System Programming
MLA	–	Microchip Libraries for Applications
MSSP	–	Master Synchronous Serial Port
OS	–	Operační systém
PIC	–	Periferal Interface Controller
PWM	–	Pulse-Width Modulation
RNDIS	–	Network Driver Interface Specification
USB	–	Universal Serial Bus

Seznam obrázků

Seznam tabulek

Seznam výpisů zdrojového kódu

Úvod

Tématem bakalářské práce je portace programátoru mikrokontrolérů PIC Usbpicprog, založeném také na mikrokontroléru PIC. Cílem práce je portace programátoru na novější verzi mikrokontroléru a záměna komunikačního protokolu USB využívající knihovnu libusb za USB CDC, které ke komunikaci využívá sériovou komunikaci.

Usbpicprog je otevřený projekt uceleného řešení programátoru a softwaru pro stolní počítač, což znamená, že zdrojový kód včetně dokumentace a schéma zapojení jsou volně dostupné. Aktuální verze programátoru je založena na mikrokontroléru PIC18F2550, který v současném stavu řešení naráží na své limity, především programové paměti a přidání podpory pro nové mikrokontroléry je možné pouze s využitím optimalizačního kompilátoru. Při záměně komunikačního protokolu s využitím novější verze MLA je téměř nemožné umístit do paměti kompletní firmware programátoru. Z toho důvodu je logické využít modernější mikrokontrolér PIC24 s minimem úprav ve zdrojovém kódu. Řada 24 je 16bitová, má k dipozici větší paměti a disponuje dostatečně přesným vnitřním generátorem kmitočtu, takže i zapojení samotného programátoru dozná určitých změn, minimálně v oblasti zapojení externího krystalu.

Další důležitá část, které se práce věnuje jsou komunikační protokoly využívající sběrnici USB. Původní projekt Usbpicprog využívá knihovnu libusb. Libusb je multiplatformní uživatelská knihovna v jazyce C pro přístup k USB zařízením. Tuto knihovnu je vhodné nahradit, protože je to knihovna takzvaně třetí strany a není přímo vázaná k portovanému projektu. V případě ukončení podpory by byla otázka času, kdy bude třeba řešit náhradu. Z toho důvodu je lepší využít sériovou komunikaci, která je podporována operačními systémy od dob svého vzniku, práce s ní je jednodušší a pro potřeby komunikace v rámci projektu je dostačující.

Mikrokontroléry PIC

Jednočipový počítač, neboli mikrokontrolér je integrovaný obvod, který obsahuje vše potřebné pro chod aplikace bez toho, aby potřeboval další podpůrné odvody. Hlavní rozdíl mezi mikroprocesorem a mikrokontrolérem je, že mikrokontrolér má ve své architektuře zahrnutou paměť pro uložení programu (ROM, EEPROM, nebo FLASH) a paměť RAM. Navíc obsahuje další integrované bloky, například pro sériové linky, blok pro generování kmitočtu, nebo analogové a logické výstupy a vstupy, které zvyšují možnosti jeho využití.

Microchip Technology je firma, která sídlí v USA a je výrobcem mikrokontrolérů PIC, jejich čipy jsou z pravidla typu RISC a jsou založeny na harvardské architektuře, to znamená, že datová a programová paměť jsou odděleny a mají redukovanou instrukční sadu (provádění jedné instrukce trvá vždy jeden strojový cyklus a délka všech instrukcí je stejná)[6].

Práce se zabývá využitím 2 typů mikrokontrolérů, původní projekt Usbpicprog využívá PIC18F2550, který patří do rodiny PIC18F2455/2550/4455/4550 a cílem práce je jeho nahrazení novějším mikrokontrolérem PIC24F, konkrétně je použit PIC24FJ64GB002, který patří do rodiny PIC24FJ64GB004.

Tento mikrokontrolér byl vybrán, protože byl nejvhodnější z hlediska potřeb projektu a při výběru bylo nutné zohlednit tato kritéria:

1. **pouzdro** – nejdůležitějším kritériem byl výběr pouzdra DIP, které umožňuje zapojení do nepájivého kontaktního pole a praktickou realizaci projektu, což je první bod zadání této práce,

2. **USB** – dalším důležitým kritériem byla přítomnost USB komunikačního modulu, který v kombinaci s DIP pouzdrem, mají z požadovaných PIC24F pouze mikrokontroléry, které mají v názvu označení GB,

3. **programová paměť** – posledním důležitým parametrem, který rozhodoval o výběru mikrokontroléru byla velikost programové paměti, v původním projektu bylo využito mikrokontroléru s pamětí 32 kB, proto bylo vhodné v novém mikrokontroléru použít paměť alespoň 64 kB a větší.

Po aplikování těchto kritérií na výběr mikrokontroléru, zůstaly k dispozici 3 modely. Protože Usbpicprog je volně dostupným projektem, byla zohledněna i cena jednotlivých modelů a byl vybrán nejlevnější model, který odpovídá všem kritériím výběru. Další parametry vyžadované pro realizaci projektu jako vnitřní oscilátor, A/D převodník, alespoň 2 16bitové časovače a zachování počtu pinů u mikrokontroléru už byly po této selekci splněny automaticky, protože je obsahovaly všechny modely.

PIC18F2550

Model 2550 je 8bitový 28pinový mikrokontrolér s podporou USB 2.0 a USART rozhraní pro komunikaci, je možné ho napájet napětím od 2 V v závislosti na frekvenci na které bude mikrokontrolér provozován. Dále obsahuje 10bitový A/D převodník, 4 časovače, 19 zdrojů přerušení, které jsou rozděleny do dvou skupin na přerušení vysoké a nízké priority. Písmeno F v označení znamená, že je na čipu přítomna paměť FLASH, která zaručuje vícenásobnou programovatelnost. Určitou nevýhodou použitého mikrokontroléru je nutnost využití externího generátoru kmitočtu pro USB komunikační modul z důvodu velké nepřesnosti vnitřního oscilátoru. V neposlední řadě mikrokontrolér obsahuje 2 16bitové moduly CCP, které slouží pro zachycení, nebo porovnání hodnot, případně generování PWM a MSSP modul, který je využitelný pro komunikaci s dalšími periferiemi, nebo mikrokontroléry. Master SSP může pracovat ve dvou režimech a to SPI a I^2C.

Základní parametry PIC18F2550[1]:

- napájecí napětí: 2,0 - 5,5 V,
- frekvence: 4 - 48 MHz,
- flash pamět: 32 kB,
- 24 I/O pinů: 3 porty (z toho 10 pinů pro A/D převodník),
- 8/16bitových časovačů: 1/3,
- 10bitový A/D převodník,
- EUSART, USB,
- 2 externí a 1 vnitřní oscilátor.

PIC24FJ64GB002

Model mikrokontroléru z řady PIC24F použitý pro nahrazení staršího mikrokontroléru PIC18 je taktéž 28 pinový, ale je nízkonapěťový, tzn. že může být napájený maximálně 3,6 V, což způsobí malé potíže v zapojení, ale tímto výčet jeho nevýhod oproti použitému mikrokontroléru PIC18 končí.

Model PIC24FJ64GB002 je 16bitový z čehož plyne několik výhod, jako např. 16 bitů pro data a až 24 bitů pro adresu, nebo adresování až 12 MB paměťového prostoru. Taktéž je přítomna větší paměť flash, která již byla potřeba z důvodu dalšího rozšiřování podpory pro nová zařízení z portfolia Microchipu a dále z důvodu většího množství kódu pro USB CDC. Mimo jiné mikrokontrolér obsahuje až 5 časovačů, 10bitový A/D převodník, tabulku vektorů přerušení, která je velkým přínosem pro obsluhu přerušení, dále vnitřní oscilátor, který je dostatečně

přesný i pro jeho využití jako generátor kmitočtu pro USB komunikační modul.

Základní parametry PIC24FJ64GB002[3]:

- napájecí napětí: 2,0 - 3,6 V,
- frekvence: 4 - 48 MHz,
- flash pamět: 64 kB,
- 19 I/O pinů: 2 porty (z toho 9 pinů pro A/D převodník),
- 16bitových časovačů: 5,
- EUSART, USB,
- 2 externí a 2 vnitřní oscilátory.

In-Circuit Serial Programming

In-System Programming (ISP) je technika programování programovatelného zařízení po umístění zařízení do plošného spoje. In-Circuit Serial Programming je protokol pro sériové programování vyvinutý firmou Microchip, jedná se o vylepšenou verzi programovací techniky ISP implementovanou do zařízení firmy Microchip.

Existují 2 typy ICSP:

- High Voltage Programming (HVP).
- Low Voltage Programming (LVP).

První z nich využívá 2 piny pro sériovou komunikaci, PGD jako vstupně/výstupní pin pro přenos dat a PGC jako zdroj hodinového signálu, který je pouze vstupní, dále jsou využity piny napájení (V_{DD}), zem (V_{SS}) a programovací napětí (V_{PP}). Programovací napětí musí být připojeno k MCLR pinu nebo k V_{PP} pinu, který je u některých mikrokontrolérů PIC k dispozici. Pro uvedení mikrokontroléru do programovacího režimu, musí být úroveň programovacího napětí ve specifikovaném rozsahu, tento rozsah není vždy stejný, může dosahovat úrovně až 13,5 V a neexistuje žádná úroveň napětí, která by platila pro všechny mikrokontroléry PIC.

Druhý způsob programování, označovaný jako LVP, je méně obvyklý, ale kromě 2 rozdílů je stejný jako HVP:

1. programovací napětí je stejné jako napájecí napětí,
2. pro programování je použit PGM pin.

PGM pin se běžně vyskytuje na komerčně vyráběných ICSP programátorech, ale přínosem použití režimu LVP je, že je možné naprogramovat několik mikrokontrolérů na desce, bez nutnosti programovat každý z nich individuálně. Nevýhodou tohoto způsobu programovaní je, že se do programovacího/ověřovacího režimu přivádí předem danou posloupností běžných logických úrovní a je těžké zabezpečit, aby tato předem daná posloupnost nepřišla během normálního provozu.

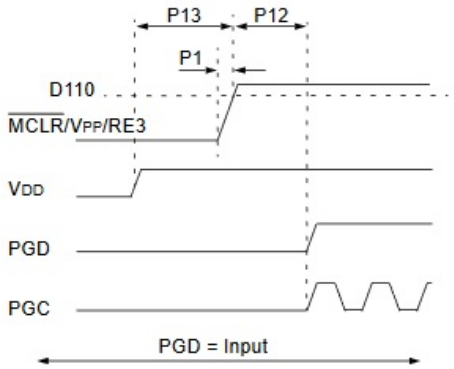
Usbpicprog programátor využívá pouze programování pomocí HVP a proto se práce bude dále zabývat pouze tímto způsobem programování.

Programovací/ověřovací režim

Abychom programované zařízení přepnuli do programovacího/ověřovacího režimu, je nejdříve nutné provést reset zařízení. Piny PGD a PGC uvést na nízkou úroveň napětí, napětí na pinu V_{PP} zvýšit na programovací napětí a zároveň napětí na V_{DD} musí být takové, aby umožnilo programování. To znamená, že napětí na V_{DD} musí být stejné, jako je vysoká úroveň napětí na PGD a PGC. I když je mikrokontrolér možné napájet až 5 V, ale napětí na PGD a PGC

je maximálně 3,3 V, je nutné tomu napětí na V_{DD} přizpůsobit. Tento proces je znázorněn na obrázku 1.

U jednotlivých řad mikrokontrolérů se tato sekvence může mírně lišit, například odesláním specifického klíče předtím, než se zvýší programovací napětí.



Obrázek 1: Vstup PIC18F do programovacího/ověřovacího režimu[2]

Během ICSP je Watchdog Timer (WDT) vypnutý, aby negeneroval neočekávaný reset zařízení.

Usbpicprog

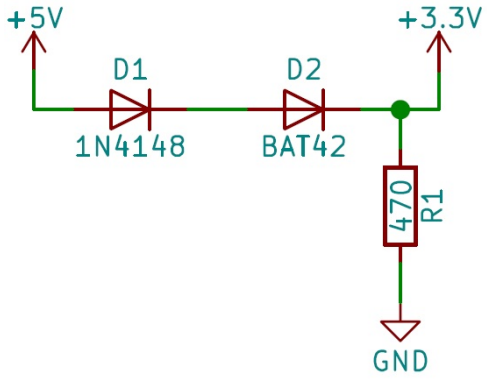
Usbpicprog je otevřené, volně dostupné řešení programátoru pro mikrokontroléry PIC. Toto řešení bylo představeno 7. února 2009 Fransem Schreuderem na FOSDEM (Free and Open Source Developers European Meeting[7] a od té doby běží jeho vývoj a skládá se ze 3 hlavních částí:

- **Hardware** – návrh plošného spoje, který obsahuje nezbytné komponenty pro propojení portu USB a konektoru ICSP s programátorem.
- **Firmware** – software běžící na mikrokontroléru, obsahuje libusb a programovací algoritmy pro podporované zařízení PIC.
- **PC software** – aplikace běžící na PC založená na wxWidgets, která slouží ke komunikaci s firmwarem.

Hardware

Regulátor napětí

Záměnou mikrokontroléru sloužícího jako programátor a především změnou napájecího napětí u tohoto mikrokontroléru bylo nutné udělat několik změn v zapojení. Především původní mikrokontrolér PIC18 bylo možné napájet 5 V, tudíž napětím dodávaným z USB portu. Nový mikrokontrolér PIC24F je možné napájet maximálně 3,6 V, proto bylo nutné napájecí napětí USB snížit vhodným regulátorem napětí dle obrázku 2, toto jednoduché řešení je jedno z mnoha doporučených v datasheetu Microchipu[5].

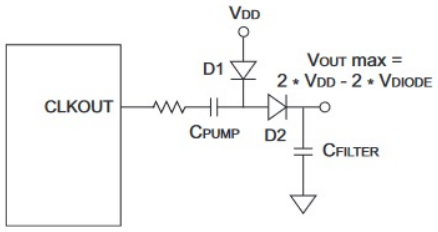


Obrázek 2: Diodový regulátor napětí[5]

Tento problém byl vyřešen zapojením diody 1N4148 a Shotkyho diody BAT42 do série v propustném směru, tím bylo dosaženo napětí 3,4 V na výstupu z regulátoru, samotný mikrokontrolér je pak napájený 3 V.

Nábojová pumpa

V původním projektu je využito tzv. nábojové pumpy, příklad z datasheetu[5] je znázorněn na obrázku 3. Toto řešení je zde popsáno, jako jednoduché řešení, jak zvýšit úroveň napětí. Lze takto zapojit i několik kondenzátorů C_{PUMP} a diod D2 do série a díky tomu zvýšit napětí na více než dvojnásobek napájecího napětí, samozřejmě je nutné počítat s úbytky napětí na PN přechodech diod. Zapojením několika pump dochází k násobení frekvence, proto je nutné nakonec zapojit jeden kondenzátor jako filtr, který je na obrázku 3 znázorněn jako C_{FILTER}.



Obrázek 3: Příklad nábojové pumpy 1. úrovně[5]

Toto řešení bylo v práci nahrazeno jednodušším DC/DC měničem napětí z 5 V na 12 V a bylo zvoleno z důvodu jednoduchosti zapojení, protože využitím nábojové pumpy pro zvýšení napětí u PIC24F došlo k obsazení příliš mnoha pinů mikrokontroléru.

Zapojení mikrokontroléru

Poslední část, kde bylo nutné udělat změny, co se týče zapojení, bylo mapování jednotlivých funkcí programátoru na piny mikrokontroléru. Mikrokontrolér PIC24F potřebuje pro svou funkci více pinů, ale díky zjednodušení zapojení využitím napěťového měniče, nebyl žádný problém tyto změny realizovat. Obsazení pinů na použitých mikrokontrolérech a jejich porovnání lze vidět v tabulce 1. Zde je možné si všimnout, že i když zapojení pinů je jiné, tak jak rozhraní USB, tak USB CDC využívají stejné piny a liší se pouze v programové části, která je popsána dále.

Piny označené _SELF jsou určené k programování samotného programátoru pomocí ICSP. V_{DD} slouží k napájení mikrokontroléru a V_{SS} slouží jako zem. V_{USB} slouží k napájení USB modulu a V_{BUS} slouží k přivedení napětí 5 V na USB v režimu hostitele. DISVREG a V_{CAP} slouží k nastavení napěťového regulátoru. D+ a D- jsou datové piny USB. Piny Pump 1 a Pump 2 nejsou u nového mikrokontroléru využity, ale původně sloužili pro zapojení nábojové pumpy. PGD a PGC piny jsou vlastní ICSP programovací piny programátoru. Piny s označením _CTL slouží k ovládání programovacího a napájecího napětí u programovaného mikrokontroléru a ADC_INPUT slouží jako vstup pro A/D převodník, který měří hodnotu programovacího napětí a tuto hodnotu předává aplikaci, která ji zobrazí uživateli.

Na obrázku 4 je možné vidět nové zapojení, přizpůsobené novému mikrokontroléru, včetně všech výše popsaných změn a pro porovnání na obrázku 5 lze nalézt zapojení původního projektu Usbpicprog před změnami.

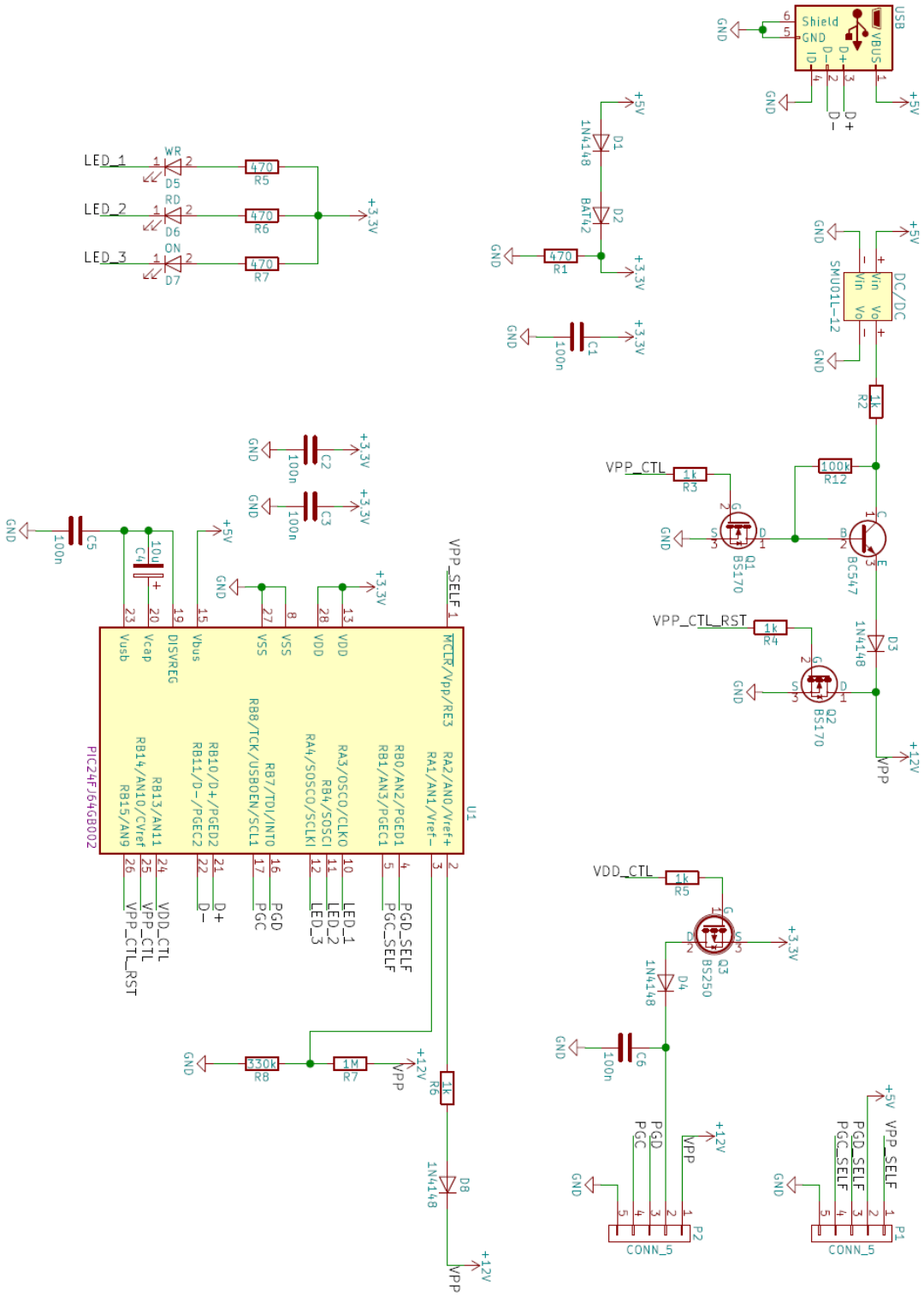
Piny	PIC18F2550	PIC24FJ64GB002
VPP_SELF	1	1
PGC_SELF	27	5
PGD_SELF	28	4
V_{DD}	20	13, 28
V_{SS}	8, 19	8, 27
V_{USB}	14	23
V_{BUS}	-	15
DISVREG	-	19
V_{CAP}	-	20
D+	16	21
D-	15	22
Pump 1	3	-
Pump 2	2	-
LED 1	11	10
LED 2	12	11
LED 3	13	12
PGD	21	16
PGD_LOW	6	-
PGC	22	17
PGC_LOW	7	-
VDD_CTL	23	24
VPP_CTL	24	25
VPP_CTL_RST	26	26
VPP_RUN	5	2
ADC_INPUT	4	3

Tabulka 1: Rozvržení funkcí na pinech mikrokontroléru

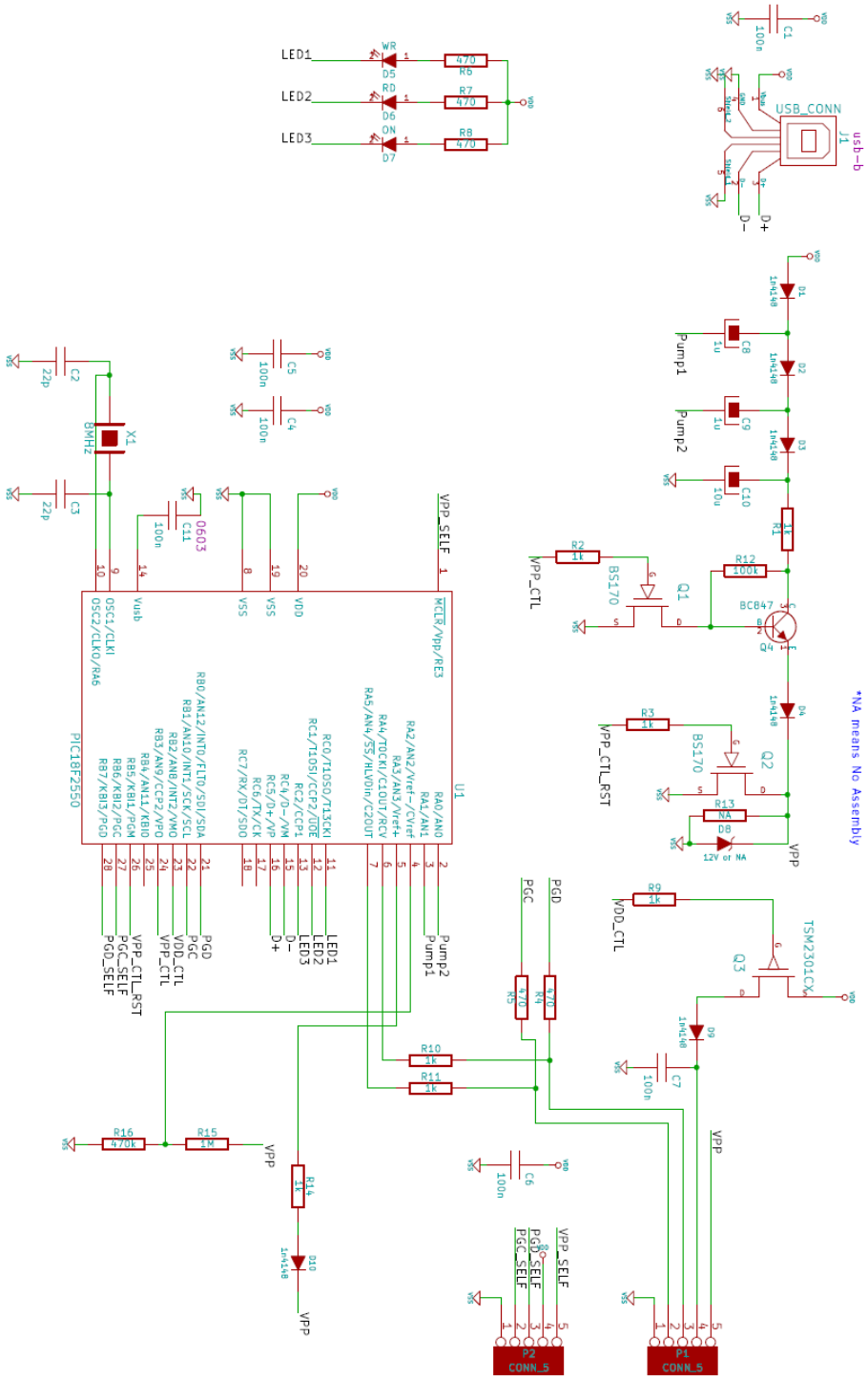
Piny označené _LOW sloužily pro snížení napětí na programovacích pinech při programování nízkonapěťových mikrokontrolérů. Vzhledem k tomu, že použitý model PIC24F je nízkonapěťový, tak tyto piny nejsou v portovaném projektu obsazeny. Naopak zde vznikl problém, jak programovat zařízení, které není možné programovat nízkým napětím. Jednou možností jak tento problém řešit je, že pro napájení takového zařízení využijeme napěťového děliče, připojeného na V_{BUS} samotného USB konektoru a 5 V tolerantní pin mikrokontroléru PIC24F. Výstupem tohoto děliče by bylo napětí 5 V v případě, že 5 V tolerantní piny budou jako nastaveny jako výstupní, případně 3,3 V v případě, že budou jako vstupní. Za tímto účelem zůstal neobsazen pin 18.

U pinů PGD_LOW a PGC_LOW je tento problém možné řešit tzv. MOSFET Translatorem, znázorněn na obrázku 6, který lze nalézt v datasheetu Microchipu[5].

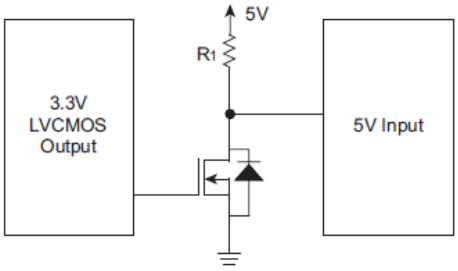
Závěrem této části je na obrázku 7 zobrazena praktická realizace programátoru Usbpicprog s programovaným mikrokontrolérem. Programátor samotný se nachází vlevo a v pravé části se nachází mikrokontrolér PIC18F2550, který sloužil otestování funkčnosti programátoru.



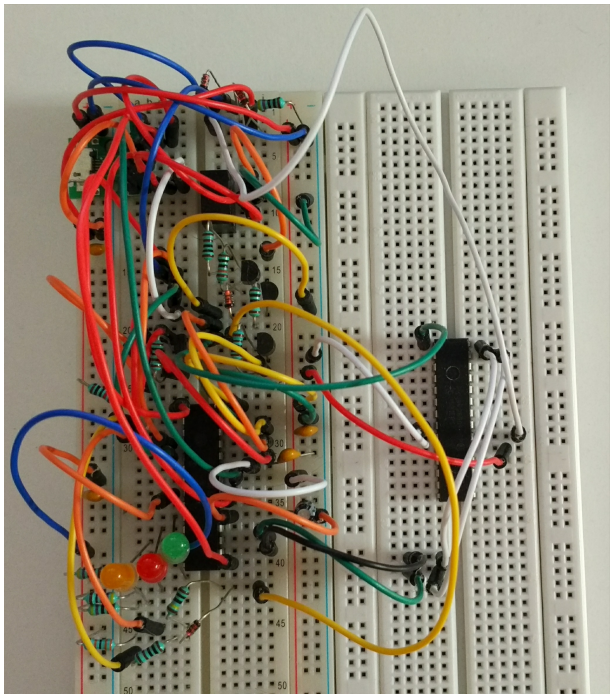
Obrázek 4: Schéma zapojení pro PIC24F



Obrázek 5: Schéma původního projektu pro PIC18[8]



Obrázek 6: MOSFET Translator[5]



Obrázek 7: Praktické zapojení Usbpicprog s PIC24F

Firmware

Firmware je napsán v jazyce C a pro úpravu zdrojového kódu bylo využito prostředí MPLAB od firmy Microchip. Pro PIC24F byl použit kompilátor XC16, který je navržen pro podporu 16bitových zařízení. Tento kompilátor je možné využívat se základními optimalizacemi překladu, i přesto zdrojový kód zaplní přibližně 2/3 programové paměti. S využitím optimalizací placených licencí je možné ušetřit další místo pro rozšiřování funkčnosti a podpory pro více zařízení.

Projekt firmwaru se skládá z přibližně 50 souborů se zdrojovým kódem a hlavičkových souborů. Zde jsou zmíněny pouze soubory definující funkcionalitu.

USB Communication Device Class

CDC je označení pro třídu zařízení, která jsou připojená pomocí rozhraní USB, ale k vlastní komunikaci používají sériovou komunikaci. USB CDC bylo původně určeno pro síťová zařízení a Microsoft se snažil tento rozhraní nahradit vlastním derivátem nazývaným Microsoft RNDIS, ale od Windows Vista podporuje i USB CDC. V dnešní době je toto rozhraní využíváno převážně v průmyslovém prostředí, kvůli udržení softwarové kompatibility a využívají ho výrobci mikrokontrolérů z důvodu snazšího vývoje USB <–> RS-232 zařízení a další využití je také v oblasti mobilních telefonů.

Konfigurační bity

První věcí, kterou je nutné nastavit při zapojení nového mikrokontroléru jsou konfigurační bity. Kvůli jejich nastavení je potřeba se zorientovat v datasheetu[3] pro daný mikrokontrolér, zde jsou mimo jiné uvedeny všechny volby pro nastavení bitů, ale klíčovou roli v této práci hrály konfigurační bity oscilátoru, protože bez správného nastavení frekvence nefunguje komunikace pomocí USB. Základem pro jeho nastavení je orientace ve schématu oscilátoru FIGURE 8-1, v části *„8.0 OSCILLATOR CONFIGURATION“* zobrazeném v datasheetu[3]. Ostatní konfigurační bity byly ponechány na defaultních hodnotách, případně byla funkce kterou nastavovali vypnuta. Nastavení oscilátoru portovaného projektu je uvedené ve výpise 1.

```
#pragma config POSCMOD = NONE // Primary Oscillator disabled
#pragma config I2C1SEL = PRI // Use default SCL1/SDA1 pins for I2C1
#pragma config IOL1WAY = OFF // The IOLOCK bit can be set and cleared using the
    unlock sequence
#pragma config OSCIOFNC = ON // OSCO pin functions as port I/O (RA3)
#pragma config FCKSM = CSECME // Sw Enabled, Mon Enabled
#pragma config FNOSC = FRCPLL // Fast RC Oscillator with Postscaler and PLL
   module (FRCPLL)
#pragma config PLL96MHZ = ON // 96 MHz PLL Startup is enabled automatically on
   start-up
#pragma config PLLDIV = NODIV // Oscillator input divided by 1 (4 MHz input)
#pragma config IESO = OFF // IESO mode (Two-Speed Start-up) disabled
```

Výpis 1: Konfigurační bity oscilátoru pro PIC24F

Časovače

Neméně důležitou částí jsou časovače, kterých je v projektu využíváno. Nastavení časovačů podle datasheetu[3], je poměrně snadnou záležitostí. Jejich detailní nastavení je rozvedeno v referenčním datasheetu pro časovače PIC24F[4], ale pro nastavení je nejdůležitější si ze schématu povšimnou, jak daný mikrokontrolér obsluhuje časovač. Zatím co PIC18 generuje přerušení při

přetečení čítacího registru a je tedy nutné tento registr při každém přerušení znova nastavit v případě, že jsou požadována přerušení po určitém časovém úseku. PIC24F využívá komparátoru, který hodnotu čítacího registru porovnává s hodnotou registru nastaveného při inicializaci časovače a pokud jsou tyto hodnoty shodné, tak generuje přerušení. Pro výpočet hodnoty, kterou je potřeba nastavit do registru slouží vzorec 1:

$$Value = \frac{Xseconds * F_{OSC} * Prescaler}{2} \tag{1}$$

- Value – výsledná hodnota, kterou je třeba nastavit do příslušného registru.
- Xseconds – požadovaný časový úsek v sekundách.
- F_{OSC} – frekvence mikrokontroléru.
- Prescaler – před-dělička, jejíž hodnota je zvolena z dostupných možností podle datasheetu[3].

I/O piny

Co mikrokontroléru umožňuje komunikovat se svým okolím jsou jeho piny, kterým je možné přiřadit různé funkce podle potřeb projektu. Výpis 2 odpovídá přiřazení funkcí zobrazených v tabulce 1.

```
#define VPP LATBbits.LATB14
#define TRISVPP TRISBbits.TRISB14
#define VPP_RST LATBbits.LATB15
#define TRISVPP_RST TRISBbits.TRISB15
#define PGD LATBbits.LATB7
#define TRISPGD TRISBbits.TRISB7
#define PGD_READ PORTBbits.RB7
#define PGC LATBbits.LATB8
#define TRISPGC TRISBbits.TRISB8
#define VDD LATBbits.LATB13
#define TRISVDD TRISBbits.TRISB13
#define VPP_RUN LATAbits.LATA0
#define TRISVPP_RUN TRISAbits.TRISA0
#define PGD_LOW LATBbits.LATB2
#define TRISPGD_LOW TRISBbits.TRISB2
#define PGC_LOW LATBbits.LATB3
#define TRISPGC_LOW TRISBbits.TRISB3
```

Výpis 2: Piny programátoru

A/D převodník

Další část, která prošla kompletní změnou je A/D převodník, který zaznamenává úroveň programovacího napětí V_{PP}. I když A/D převodník implementovaný v PIC24F má mnohem bohatší nastavení a je výkonnější, tak po důkladném prostudování datasheetu[3] je práce s ním jednodušší. Hlavní předností převodníku použitého u PIC24F je možnost přepnout pouze jeden pin do analogového režimu při inicializaci a dál s jeho nastavením nic nedělat, u PIC18 bylo nutné neustále přepínat určité množství pinů mezi analogovým a digitálním režimem, aby bylo možné používat převodník i další funkce umístěné na okolních pinech. Inicializace převodníku PIC24F použitá v portovaném projektu je zobrazená ve výpisu 3.

```
void InitAdc( void ) {
    AD1CON1 = 0x00E0; // internal counter ends sampling and starts converting
    AD1CON3 = 0x1F02; // 31 Tad, Tad = 3Tcy
    AD1CHS = 0x0001; // Input on AN1
    AD1PCFG = 0xFFFD; // Pin AN1 to analogue
    TRISAbits.TRISA1 = 1; // Input;
    AD1CON1bits.ADON = 1; // Turn on ADC
}
```

Výpis 3: Inicializace A/D převodníku PIC24F

Funkce čtení naměřených hodnot zobrazená ve výpisu 4 je díky odstraněným problémům s přepínáním režimů na pinech jednodušší a jediný rozdíl je dán velikostí registru kam se hodnoty zaznamenávají. Po každém čtení je nutné hodnotu z 16bitového registru ADC1BUF0 rozdělit pomocí funkce bitwise a bitového posunu do dvou 8bitových proměnných.

```
void ReadAdc( unsigned char* data ) {
   AD1CON1bits.SAMP = 1; // start sampling input
    while(!AD1CON1bits.DONE){};

    data[0] = ADC1BUF0 & 0xFF;
    data[1] = ADC1BUF0 >> 8;
}
```

Výpis 4: Čtení hodnot z A/D převodníku PIC24F

Komunikace firmwaru a aplikace

O inicializaci USB CDC se stará firmware programátoru a v PC se programátor hlásí jako CDC ACM, což je emulovaný sériový port přes USB rozhraní. Aplikace před začátkem komunikace provede nezbytnou inicializaci používaného COM portu. Nutno zmínit, že byť USB CDC používá komunikaci pomocí sériové komunikace přes USB, tak v mikrokontroléru nedochází k předávání

dat do UART, místo toho jsou data po přečtení z USB CDC okamžitě zpracována. Jako základ pro realizaci komunikace pomocí USB CDC, byl využit ukázkový projekt Serial Emulatoru dostupný v MLA, který je velice dobře zpracovaný a k jeho uvedení do funkční podoby v rámci portovaného projektu stačilo pouze nahradit soubory pro USB rozhraní.

Soubory zajišťující komunikaci

- **usb_hal_pic24.c** – soubor obsahující funkce pro abstrakci rozhraní hardwaru, lze ho využít pro práci na rozdílných mikrokontrolérech jako PIC18 nebo PIC24,
- **usb_function_cdc.c** – soubor obsahující všechny funkce, makra, definice, proměnné, datové typy atd. které jsou potřebné pro komunikaci s USB CDC ovladačem, obsahuje funkce inicializace, čtení nebo zápisu a obsluhu USB CDC,
- **usb_device.c** – soubor obsahující funkce, makra, definice, proměnné, datové typy, atd. pro obsluhu USB zařízení, která používají zásobník.

Čtení dat z USB CDC

Čtení dat z USB rozhraní v původním projektu zajišťovala funkce USBGenRead() a ta byla v portovaném projektu pro CDC nahrazena funkcí getsUSBUSART(), uvedenou ve výpise 5. Tato funkce zkopíruje řetězec bajtů na uživatelem specifikovanou pozici. Čtení je neblokující funkce, která nečeká pokud nemá žádná data ke čtení. Místo toho vrátí '0' na znamení, že žádná data nejsou k dispozici. Nevýhodou této funkce je, že množství přijatých dat musí být menší než velikost bufferu pro příchozí data. Nicméně o tom kolik dat se odešle a jak budou rozdělena rozhoduje USB CDC driver hostitele.

```
BYTE getsUSBUSART(unsigned char *buffer, BYTE len) {
    cdc_rx_len = 0;

    if(!USBHandleBusy(CDCDataOutHandle)) {

        // Adjust the expected number of BYTEs to equal the actual number of
            BYTEs received.
        if(len > USBHandleGetLength(CDCDataOutHandle))
            len = USBHandleGetLength(CDCDataOutHandle);

        // Copy data from dual-ram buffer to user's buffer
        USBMaskInterrupts();
        for(cdc_rx_len = 0; cdc_rx_len < len; cdc_rx_len++)
            buffer[cdc_rx_len] = cdc_data_rx[cdc_rx_len];
```

```
        // Prepare dual-ram buffer for next OUT transaction
        CDCDataOutHandle = USBRxOnePacket(CDC_DATA_EP,(BYTE*)&cdc_data_rx,
            sizeof(cdc_data_rx));
        USBUnmaskInterrupts();
    }
    return cdc_rx_len;
}
```

Výpis 5: Funkce pro čtení dat z USB

Odesílání dat do USB CDC

Před pokusem o odeslání dat je nejdříve nutné uživatelem zkontrolovat zda cdc_trf_state = CDC_TX_READY, k tomu pro USB CDC slouží předdefinované makro USBUSARTIsTxTrfReady(), použití tohoto makra v rámci projektu je možné vidět ve výpisu 6, toto makro musí vrátit hodnotu TRUE.

```
if(mUSBUSARTIsTxTrfReady()) {
            putUSBUSART((unsigned char *) &output_buffer[0], counter);
            counter = 0;
}
```

Výpis 6: Ukázka kontroly a volání funkce putUSBUSART()

Zápis dat do USB CDC je realizován funkcí putUSBUSART() znázorněnou ve výpisu 7, která zapíše pole dat do USB rozhraní. Přenos dat od zařízení k hostiteli je mnohem flexibilnější, než přenos od hostitele do zařízení, jelikož je tímto způsobem možné přenést větší množství dat rozdělením do více transakcí, ale je třeba periodicky volat funkci CDCTxService().

```
void putUSBUSART(unsigned char *data, BYTE length) {
    USBMaskInterrupts();

    if(cdc_trf_state == CDC_TX_READY) {
        mUSBUSARTTxRam((BYTE*)data, length);
    }

    USBUnmaskInterrupts();
}
```

Výpis 7: Funkce pro zápis dat do USB

Funkce, nebo spíše služba CDCTxService(), zpracovává transakce mezi zařízením a hostitelem a je také hlavní rozdílem v komunikaci oproti původnímu projektu. Tato služba by měla

být volána jednou na konci smyčky hlavního programu, po dosažení nakonfigurovaného stavu zařízení (po dokončení funkce CDCInitEP(), kdy USBDeviceState = CONFIGURED_STATE a cdc_trf_state = CDC_TX_READY). Služba se stará o rozdělení přenášených dat do více transakcí před zasláním k hostiteli ve formě sériových dat. Nevoláním této služby periodicky zabráníme odesílání dat k hostiteli. Ve výpise 8 je uveden pouze příklad použití této služby, celý zdrojový kód lze najít v příloze A.

```
void main(void) {
    USBDeviceInit();
    while(1) {
        USBDeviceTasks();
        if((USBGetDeviceState() < CONFIGURED_STATE) ||
            (USBIsDeviceSuspended() == TRUE)) {
            // Either the device is not configured or we are suspended so we
            // don't want to do execute any application code
            continue; // go back to the top of the while loop
        }
        else {
            // Run application code.
            UserApplication();

            // Keep trying to send data to the PC as required
            CDCTxService();
        }
    }
}
```

Výpis 8: Příklad služby obsluhy přenosu dat ze zařízení k hostiteli

Soubory programátoru:

- **bulk_erase.c** – soubor obsahující funkce k vymazání pamětí a konfigurace mikrokontroléru,
- **device.c** – soubor, který udržuje seznamy rodin a typy podporovaných zařízení a aktuálně používaný mikrokontrolér,
- **prog_lolvl.c** – soubor obsahující funkce obsluhy vstupu zařízení do režimu programování/oveření a jejich ukončení ,
- **read_code.c / read_data.c** – soubory, které obsahují funkce čtení programové a datové paměti,

- **write_code.c / write_data.c / write_config_bits.c** – soubory, které obsahují funkce zápisu do programové a datové paměti a zápis konfiguračních bitů mikrokontroléru.

Posledním důležitý soubor firmwaru je **main.c**, který obsahuje hodnoty konfiguračních registrů, dále funkci volání inicializace a smyčku hlavního programu, kde probíhá zpracování přijatých dat zakončené voláním funkce CDCTxService(), vše popsané výše.

Programovací/ověřovací režim

Zde je ve výpisu 9 vyjádřen obrázek 1 zdrojovým kódem. Tento zdrojový kód slouží ke vstupu zařízení PIC18 do programovacího/ověřovacího režimu. Tuto funkci obsahuje soubor **prog_lolvl.c**.

Vstup do programovacího/ověřovacího režimu u PIC18 je zde vybrán jako příklad pro svou jednoduchost. Vstup do tohoto režimu u jiných mikrokontrolérů PIC je obdobný a liší se například dodatečným odesláním posloupnosti bitů.

```
void enter_ISCP_simple() {
   enablePGC_D();
   PGDlow();
   PGClow();

   clock_delay();
   VDDon();
   DelayMs( 100 );
   VPPon();
   DelayMs( 100 );
}
```

Výpis 9: Funkce vstupu zařízení PIC18F do programovacího/ověřovacího režimu

GUI rozhraní pro PC

Aplikace pro PC slouží uživateli jako ovládací prvek celého projektu, který zasílá příkazy firmwaru pomocí několikrát zmiňovaného USB CDC, který je vykoná a výsledek vrátí aplikaci. Celá aplikace je napsána v C++ a je možné ji ovládat pomocí příkazové řádky nebo grafického rozhraní založeného na wxWidgets.

wxWidgets

Je otevřená knihovna vytvořená v C++ určená pro tvorbu aplikací na systémech Windows, Mac OS X, Linux aj. na společné kódové základně společné pro všechny systémy. Grafické API wxWidgets původně začalo jako projekt pro tvorbu aplikací, které mají být přenositelné mezi Unix a Windows systémem, později byla přidána podpora pro Mac OS X, GTK+ a další. Jak již

bylo zmíněno, hlavní výhodou je použitelnost této knihovny pro více systémů, čehož je v rámci projektu Usbpicprog využíváno. Knihovna samotná slouží jako API pro psaní GUI aplikací na různých platformách, které stále využívají ovládací prvky a utility nativní platformy. Spojí se s příslušnou knihovnou pro platformu a kompilátorem a aplikace tak má vhodný vzhled pro danou platformu.

Popis souborů aplikace:

Zde jsou opět zmíněny pouze soubory, které definují funkcionalitu projektu.

- **main.cpp** – je vstupním bodem aplikace odkud se volá sestavení grafického rozhraní a nachází se zde i inicializace parseru pro ovládání pomocí příkazové řádky,
- **uppmainwindow_base.cpp** – slouží k sestavení hlavního okna,
- **uppmainwindow.cpp** – soubor obsahuje funkce pro dokončení grafického rozhraní a přidává funkce pro vláknové operace a další,
- **hardware.cpp** – obsahuje funkce pro práci s hardwarem,
- **pictype.cpp** – zde je řešeno načítání a uchování dat ze souboru XML, který obsahuje data podporovaných mikrokontrolérů,
- **hexfile.cpp** – zajišťuje funkcionalitu pro obsah HEX souboru zobrazený v hlavním okně.

Aplikace obsahuje další hlavičkové soubory a soubory se zdrojovým kódem, například pro další okna jako nastavení aplikace nebo testování funkčnosti pinů programátoru.

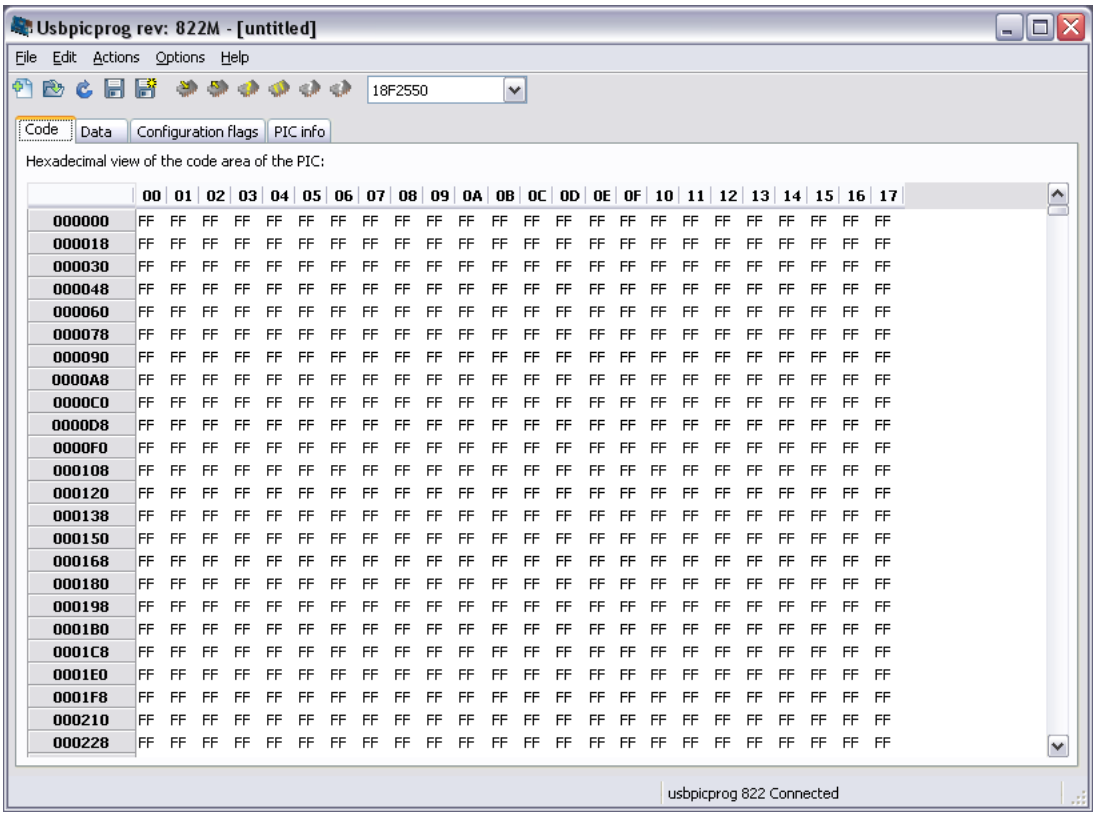
Hlavní okno aplikace

Hlavní okno obsahuje největší část funkcionality celé aplikace a je zobrazeno na obrázku 8. Největší prostor je zde věnován 4 záložkám, které jsou popsány níže. Obsah těchto záložek se mění v závislosti na detekovaném mikrokontroléru, ale je zde možnost vybrat mikrokontrolér i manuálně v ComboBoxu, který je možné vidět na konci druhého řádku.

Popis záložek hlavního okna:

- **Programová paměť (Code)** – tato záložka se při spuštění aplikace zobrazí jako vymazaná (ve všech buňkách je zobrazeno FF), po načtení dat z připojeného mikrokontroléru se zde zobrazí obsah jeho programové paměti, případně se zde zobrazí data HEX souboru načteného z disku
- **Datová paměť (Data)** – v této záložce je možné nahlédnout do datové paměti mikrokontroléru

- **Konfigurační bity (Configuration bits)** – tato záložka je rozdělena do 2 částí, v levé části je seznam dostupných registrů sloužících ke konfiguraci mikrokontroléru a ve zbytku okna jsou zobrazeny hodnoty které je možná nastavit a jejich hodnota v šestnáctkové soustavě. Po spuštění aplikace, nebo změně mikrokontroléru v ComboBoxu se zobrazí defaultní hodnoty nastavení pro vybraný mikrokontrolér
- **PIC info** – zde je možné zjistit základní informace o připojeném/vybraném zařízení, jako například rozsah napájecího napětí, frekvence jádra, nebo je zde graficky zobrazeno, které piny je nutné připojit pro programování mikrokontroléru pomocí ICSP



Obrázek 8: Hlavní okno programu v prostředí Windows

Komunikace na straně aplikace

Projekt Usbpicprog je velice rozsáhlým projektem s velkým množstvím zdrojového kódu a zpočátku bylo velice těžké se v projektu orientovat bez dokumentace. Nakonec se ukázalo, že je projekt velice dobře zpracován, takže stěžejní část práce, nahrazení knihovny libusb funkcemi pro sériovou komunikaci, se ukázalo jako poměrně snadný úkol. Libusb byla v projektu napojená pouze ve 2 místech a hlavním místem, kde byla využívána, byl soubor **hardware.cpp**, proto se práce bude dále zabývat komentářem zde prováděných změn.

Aby bylo možné komunikovat se zařízením s využitím USB CDC, je třeba na straně aplikace udržovat HANDLE sériového portu využívaného ke komunikaci, který se vytvoří zavoláním funkce CreateFile() s příslušnými parametry a pro úspěšnou realizaci komunikace mezi hostem a zařízením bylo nutné nastavit struktury DCB a COMMTIMEOUTS, které jsou popsány níže. Všechny tyto činnosti pro USB v původním projektu řídila knihovna libusb.

Inicializace struktury DCB je uvedena ve výpise 10. Zapsání změněné DCB struktury pro nastavený port se provádí zavoláním funkce SetCommState();

```
DCB dcbSerialParams = { 0 };
dcbSerialParams.DCBlength = sizeof(dcbSerialParams);
dcbSerialParams.BaudRate = CBR_115200;
dcbSerialParams.ByteSize = 8;
dcbSerialParams.StopBits = ONE5STOPBITS;
dcbSerialParams.Parity = NOPARITY;
```

Výpis 10: Inicializace DCB struktury

Struktura COMMTIMEOUTS slouží jako pojistka proti nekonečnému čekání na data a zároveň dává programátoru čas na zpracování přijatého příkazu a odpověď. Použité nastavení časových prodlev je uvedeno ve výpisu 11. Pro provedení změn ve struktuře bylo nutné změny opět zapsat zavoláním funkce SetCommTimeouts().

```
COMMTIMEOUTS timeouts = { 0 };
timeouts.ReadIntervalTimeout = 0;
timeouts.ReadTotalTimeoutConstant = 500;
timeouts.ReadTotalTimeoutMultiplier = 0;
timeouts.WriteTotalTimeoutConstant = 500;
timeouts.WriteTotalTimeoutMultiplier = 0;
```

Výpis 11: Inicializace COMMTIMEOUTS struktury

Z použitého nastavení je možné vyčíst, že každá operace čtení a zápisu může trvat maximálně 500 ms, což se ukázalo v kombinaci s nastavením DCB struktury jako dostatečná časová prodleva pro přenos dat, bez znatelného vlivu na rychlost běhu aplikace.

Operace, které provádí zápis/čtení nad emulovaným sériovým portem realizovány pomocí standardních funkcí WriteFile() pro zápis, nebo ReadFile() pro čtení z využívaného portu.

Před úspěšným dokončením práce se v aplikaci pro PC vyskytly ještě dva problémy, které bylo třeba vyřešit. První z nich souvisel se změnou v zapojení, kde bylo nutné upravit dělič napětí, kvůli omezení maximálního vstupního napětí z 5 V na 3,6 V. Tento dělič upravoval hodnotu napětí přiváděného na pin A/D převodníku. Hodnota naměřená převodníkem je přímo předávána aplikaci, kde probíhá přepočet na úroveň napětí.

Druhý řešený problém byl komplikovanější a týkal se operace kontroly zda je programovaný mikrokontrolér vymazaný. V aplikaci je tato operace reprezentována funkcí blankCheck().

Operace vymazání, která je v aplikaci reprezentována funkcí bulkErase() byla vždy dokončena úspěšně, všechna data na mikrokontroléru byla smazána a konfigurační bity byly nastaveny na defaultní hodnoty. Při kontrole vymazání proběhl test pamětí v pořádku, ale test konfiguračních bitů vždy selhal. Bylo to zapříčiněno chybným voláním funkce GetMask(), která chybně nastavovala defaultní hodnoty bitů v nově vytvořeném HEX souboru, který se sloužil pro porovnání s hodnotami čtenými z mikrokontroléru. Funkce GetMask(), znázorněná ve výpise 12, prováděla bitovou operaci bitwise nad bity registru, které je možné nastavit v rámci dané části registru a tudíž její výsledek byl vždy maximální hodnota nastavení všech funkcí, které je možné v daném registru nastavit, proto se tato hodnota nikdy nemohla shodovat s očekávanou hodnotou čtenou z mikrokontroléru.

```
unsigned long GetMask() const {
    int tmpConfigMask=0;
    for (unsigned int j=0; j<Values.size(); j++)
        tmpConfigMask |= Values[j].Value;

    return tmpConfigMask;
}
```

Výpis 12: Funkce GetMask()

Nicméně řešení bylo podstatně jednodušší než pochopení tohoto problému. Všechna potřebná data o používaném mikrokontroléru jsou uložena v XML souborech přístupných aplikaci. Zde jsou i defaultní hodnoty pro jednotlivé funkce nastavované po vymazání mikrokontroléru. Při spuštění aplikace, změně mikrokontroléru, nebo automatické detekci mikrokontroléru, probíhá načtení dat z příslušného XML souboru do proměnných v aplikaci. Řešením tohoto problému bylo při načítání těchto souborů, načíst do aplikace i defaultní hodnoty, což se neprovádělo a při zavolání funkce pro vytvoření nového souboru pak vrátit pouze tyto hodnoty ne masku vytvořenou z hodnot, které je možné nastavit.

Ověření funkčnosti

Pro otestovaní funkčnosti programátoru bylo využito převážně funkcí, které projekt již obsahoval. Celý projekt včetně všech úprav byl realizován na virtuálním PC se systémem Windows 7 v prostředí VirtualBox, na hostitelském PC se systémem Ubuntu 16.04. Testování je rozděleno do 2 částí:

1. Testování spojení mezi programátorem a PC.
2. Testování ICSP mezi programátorem a programovaným mikrokontrolérem.

Test USB CDC

Základem testu funkčnosti bylo detekování programátoru s USB CDC na různých operačních systémech. Tento test byl proveden na operačních systémech:

- Ubuntu 16.04 – hostitelský systém fungoval jako základní testovací platforma a připojení na tento systém bylo otestováno více než 500x, všechny problémy, které se zde vyskytly měli původ v nekvalitním provedení desky, na které byl programátor realizován. Test opakovaného připojení programátoru 10x za sebou proběhl tak úspěšně.
- Windows 7 – pro systém provozovaný na virtuálním stroji platí to stejné co pro jeho hostitelský systém, s tím rozdílem, že se zde neprojevili problémy se zapojením, ale i zde byl proveden test opakovaného připojení programátoru 10x za sebou a také bez problémů.
- Windows 10 – pro ověření funkčnosti a kompatibility byl zvolen nejnovější systém společnosti Microsoft, kde byla testována pouze detekce programátoru a opakovaná detekce se stejnou metodikou provedení jako u předešlých 2 OS, ani zde se neprojevili žádné potíže.

```
nBytes = getsUSBUSART((unsigned char*)input_buffer, 64);

// ECHO TEST
if (nBytes != 0) {
    if(mUSBUSARTIsTxTrfReady())
        putUSBUSART((unsigned char*)input_buffer, nBytes);

    nBytes = 0;
}
```

Výpis 13: Funkce ECHO

Druhým testem v pořadí byla komunikace mezi počítačem a mikrokontrolérem programátoru. Za tímto účelem bylo využito nejdříve funkce ECHO dle výpisu 13, která četla data zaslaná přes

USB CDC a odesílala je zpět. Ke komunikaci na straně počítače bylo využito programu PuTTy, který se připojil na definovaný COM port, vše ostatní zajišťoval firmware programátoru.

Poslední test už proběhl s portovaným projektem. Při detekování a úspěšném navázání spojení mezi aplikací a firmwarem programátoru, aplikace po firmwaru požaduje informace, kterou jsou uloženy přímo v paměti mikrokontroléru. Aplikace nejdříve zasílá příkaz CMD_GET_PROTOCOL_VERSION, na který očekává odpověď číslovku 5 a poté zasílá dotaz CMD_FIRMWARE_VERSION, jako odpověď očekává řetězec „usbpicprog“ + číslo verze firmwaru. Tento řetězec se po přijetí zobrazí v pravé části stavové lišty hlavního okna aplikace. V případě, že některý z těchto příkazů firmware z jakéhokoliv důvodu neprovede, tak se v pravé části stavové lišty zobrazí hláška, že programátor nebyl nalezen.

Všechny příkazy jsou zde prezentovány ve své textové podobě, ale ve skutečnosti jsou zasílány pouze číselné hodnoty, tabulka těchto příkazů, které aplikace zasílá firmwaru je uvedena v příloze B.

Test ICSP programátoru

Další testování již probíhalo s připojeným programovaným mikrokontrolérem, v tomto případě již několikrát zmiňovaným PIC18F2550. Pro ověření funkčnosti byly využity tyto funkce programátoru:

1. Detekování modelu mikrokontroléru - provádí se jednou při spuštění aplikace a pak při kliknutí na příslušné tlačítko, pro získání informací o modelu mikrokontroléru programátor přečte data z předem známého umístění v paměti programovaného mikrokontroléru.

2. Přečtení obsahu paměti a konfiguračních bitů mikrokontroléru - přečte celou paměť programovaného mikrokontroléru a data zobrazí v záložkách hlavního okna aplikace.

3. Zapsání dat do mikrokontroléru - data v záložkách hlavního okna zapíše na příslušné místo v paměti programovaného mikrokontroléru.

4. Ověření zapsaných dat - přečte data z programovaného mikrokontroléru a porovná je z daty v záložkách hlavního okna.

5. Vymazání mikrokontroléru.

6. Kontrola vymazání mikrokontroléru - na pozadí aplikace vytvoří prázdný HEX soubor s defaultním nastavením, pro detekovaný mikrokontrolér a jeho obsah porovná s obsahem paměti programovaného mikrokontroléru.

Všechny testy nakonec proběhly úspěšně s téměř 100% pravděpodobností úspěchu i při opakovaném provedení. Všechny potíže, které se během těchto operací vyskytly jsou připisovány nekvalitnímu provedení nepájivého kontaktního pole a pohybu kontaktů.

Závěr

Práce je rozdělena do dvou částí. V první části je krátké představení mikrokontrolérů, které jsou v průběhu práce využívány a programovací technika ICSP, která je využívána portovaným programátorem.

Druhá, obsáhlejší část této práce se zabývá portovaným projektem Usbpicprog. Zde jsou dokumentovány především změny, které bylo nutné provést. Těchto změn v zásadě není mnoho, napomohlo tomu především velice dobré zpracování původního projektu, ale aby bylo možné tyto změny realizovat, bylo nutné se zorientovat v celém projektu a datasheetech využívaných mikrokontrolérů.

Velkou pozornost bylo nutné věnovat i hardwarové části projektu, která při zpracování přinesla mnohem více problémů než bylo očekáváno. A to především z důvodu rozdílu v napájení mikrokontrolérů, protože se v zapojení vyskytla další úroveň napětí, která v původním projektu nebyla. Při tvorbě zapojení upraveného pro PIC24F byla nejdříve testována možnost rozšíření nábojové pumpy, ale toto řešení se ukázalo jako neefektivní, z výše zmíněných důvodů. Několik potíží přineslo i kontaktní nepájivé pole, použité pro praktickou realizaci zapojení, které občas svým nekvalitním zpracováním způsobovalo neočekávané chování.

Při portaci firmwaru se jako největší problém ukázalo správné nastavení časovačů, které významně ovlivňují komunikaci s programovanými mikrokontroléry i PC. Menší problémy způsobil přechod mezi používanými kompilátory. Projekt sériového emulátoru dodávaného v MLA jako příklad pro USB CDC je také velice dobře zpracovaný a proto spojení s původním projektem nebylo nijak náročné, problémy se vyskytovaly pouze ve správném nastavení I/O pinů.

Portace aplikace pro PC zabrala stejné množství času jako každá z předešlých částí, i když v rámci ní je provedeno nejméně změn. Bylo to způsobeno především velkým množstvím kódu, který bylo nutné projít a funkcí, které se volají mezi sebou. Největším problémem v této části se ukázalo správné nastavení časových zpozdění při čtení a zápisu. Na závěr práce byla v projektu objevena chyba, která se v projektu vyskytovala od počátku a byla spojená s kontrolou vymazání mikrokontroléru. Tuto chybu bylo nutné odstranit pro úspěšnou a kompletní prezentaci funkčnosti, což se i podařilo.

Výsledky této práce byly nakonec zveřejněny online na stránkách github.com na adrese github.com/lk-dll/usbpicprog.

Literatura

[1] *PIC18F2455/2550/4455/4550 Data Sheet.* Chandler: Microchip Technology Inc., 2006. 430 s.
URL: <http://ww1.microchip.com/downloads/en/devicedoc/39632c.pdf> [citováno 16. listopadu 2015]

[2] *Flash Microcontroller Programming Specification.* Chandler: Microchip Technology Inc., 2010. 56 s.
URL: <http://ww1.microchip.com/downloads/en/DeviceDoc/30009622M.pdf> [citováno 12. ledna 2016]

[3] *PIC24FJ64GB004 Family Data Sheet.* Chandler: Microchip Technology Inc., 2010. 352 s.
URL: <http://ww1.microchip.com/downloads/en/DeviceDoc/39940d.pdf> [citováno 7. března 2016]

[4] *PIC24F Family Reference Manual - Section 14. Timers.* Chandler: Microchip Technology Inc., 2006. 26 s.
URL: <http://ww1.microchip.com/downloads/en/DeviceDoc/39704a.pdf> [citováno 11. března 2017]

[5] *Compiled Tips 'N Tricks Guide.* Chandler: Microchip Technology Inc., 2009. 137 s. URL: <http://ww1.microchip.com/downloads/en/DeviceDoc/01146B.pdf> [citováno 13. května 2017]

[6] microchip.com. [online]
URL: <https://www.microchip.com/about-us/> [citováno 8. června 2017].

[7] usbpicprog.org [online]
URL: <http://usbpicprog.org/> [citováno 25. června 2017]

[8] usbpicprog.org [online]
URL: <http://usbpicprog.org/?page_id=5> [citováno 25. června 2017]

Služba CDCTxService()

```
void CDCTxService(void) {
    BYTE byte_to_send;
    BYTE i;

    USBMaskInterrupts();

    CDCNotificationHandler();

    if(USBHandleBusy(CDCDataInHandle)) {
        USBUnmaskInterrupts();
        return;
    }

    //Completing stage is necessary while [ mCDCUSartTxIsBusy()==1 ]. By having
        this stage, user can always check cdc_trf_state, and not having to call
        mCDCUsartTxIsBusy() directly.
    if(cdc_trf_state == CDC_TX_COMPLETING)
        cdc_trf_state = CDC_TX_READY;

    //If CDC_TX_READY state, nothing to do, just return.
    if(cdc_trf_state == CDC_TX_READY) {
        USBUnmaskInterrupts();
        return;
    }

    //If CDC_TX_BUSY_ZLP state, send zero length packet
    if(cdc_trf_state == CDC_TX_BUSY_ZLP) {
        CDCDataInHandle = USBTxOnePacket(CDC_DATA_EP,NULL,0);
        cdc_trf_state = CDC_TX_COMPLETING;
    }
    else if(cdc_trf_state == CDC_TX_BUSY) {
        //First, have to figure out how many byte of data to send.
      if(cdc_tx_len > sizeof(cdc_data_tx))
          byte_to_send = sizeof(cdc_data_tx);
      else
          byte_to_send = cdc_tx_len;
```

```
        //Subtract the number of bytes just about to be sent from the total.
      cdc_tx_len = cdc_tx_len - byte_to_send;

        pCDCDst.bRam = (BYTE*)&cdc_data_tx; // Set destination pointer

        i = byte_to_send;
        if(cdc_mem_type == USB_EP0_ROM) { // Determine type of memory source
            while(i) {
                *pCDCDst.bRam = *pCDCSrc.bRom;
                pCDCDst.bRam++;
                pCDCSrc.bRom++;
                i--;
            }//end while(byte_to_send)
        }
        else { // _RAM

            while(i) {
                *pCDCDst.bRam = *pCDCSrc.bRam;
                pCDCDst.bRam++;
                pCDCSrc.bRam++;
                i--;
            }//end while(byte_to_send._word)
        }//end if(cdc_mem_type...)

        //Lastly, determine if a zero length packet state is necessary. See
            explanation in USB Specification 2.0: Section 5.8.3
        if(cdc_tx_len == 0) {
            if(byte_to_send == CDC_DATA_IN_EP_SIZE)
                cdc_trf_state = CDC_TX_BUSY_ZLP;
            else
                cdc_trf_state = CDC_TX_COMPLETING;
        }//end if(cdc_tx_len...)
        CDCDataInHandle = USBTxOnePacket(CDC_DATA_EP,(BYTE*)&cdc_data_tx,
            byte_to_send);
    }//end if(cdc_tx_sate == CDC_TX_BUSY)
    USBUnmaskInterrupts();
}
```

Seznam příkazů pro firmware zasílaných PC aplikací

```
typedef enum {
    CMD_ERASE = 0x10,
    CMD_READ_ID = 0x20,
    CMD_WRITE_CODE = 0x30,
    CMD_READ_CODE_OLD = 0x40, // original read code/config command
        differentiated by address (and blocksize <= 8 except for PIC18F2221 and
        PIC18's don't distinguish code/config )
    CMD_WRITE_DATA = 0x50,
    CMD_READ_DATA = 0x60,
    CMD_WRITE_CONFIG = 0x70,
    CMD_SET_PICTYPE = 0x80,
    CMD_FIRMWARE_VERSION = 0x90,
    CMD_DEBUG = 0xA0,
    CMD_GET_PIN_STATUS = 0xB0,
    CMD_SET_PIN_STATUS = 0xC0,
    CMD_GET_PROTOCOL_VERSION = 0xD0,
    CMD_READ_CODE = 0xD1,
    CMD_READ_CONFIG = 0xD2,
    CMD_EXIT_TO_BOOTLOADER = 0xD3,
    CMD_MREAD_CODE = 0xD4,
   CMD_APPLY_SETTINGS = 0xD5,
   CMD_INVALID
}CMD_UPP;

typedef enum {
   PROT_NONE, // not connected
   PROT_BOOTLOADER0, // Connected to bootloader
   PROT_UPP0 = 100, // Original command set to firmware
   PROT_UPP1, // deleted READ_CODE(0x40) added READ_CODE(0xD1),READ_CONFIG(0xD2
       ),EXIT_TO_BOOTLOADER(0xD3)
   PROT_UPP2, // added multi-block read to READ_CODE(0xD4)
   PROT_UPP3 // added configuration to limit VDD, VPP and PGD, PGC
}PROTOCOL;
```